

Hibernate by Example

*Eitan Suez,
UptoData Inc*

About the Speaker

- Java Programmer
- <http://u2d.com/>
- Weblog on <http://java.net/>
- NFJS Speaker

Goals

- To get you up and running with Hibernate
- To Learn O/R Mapping with Hibernate, in a hands-on, iterative manner
- To get a good, first-hand feel of this framework

Motivation

- My experience using Hibernate has convinced me that it has many gems, is useful in many circumstances, and is worth studying
- The belief that the best way to learn something is by doing it actively

Style

Do {

- Model a class of objects
- Construct database mapping
- Export or update database schema
- Write Hibernate code to save sample data to database
- Write Hibernate code to query database

} until we've covered most of the mapping features of Hibernate

Disclaimer

- There is a lot to this framework, cannot cover every aspect in a simple 1-2 hr course
- Emphasis on constructing a meaningful sample application at the expense of completeness: I will not be covering every minute detail of the framework

Agenda

1. Project Background
2. Mapping
3. The API
4. Session Usage Strategies
5. Performance
6. Batch Processing
7. UserType's
8. Annotations
9. Tools, Hibernate 3 features

What is Hibernate?

- An Object/Relational Mapping (O/R M) API for Java
- Open Source (LGPL)
- Today a part of RedHat
- Principal author: Gavin King
- Other Major Figure: Christian Bauer
- Almost a defacto standard O/R M for Java
- Current version 3.1 (3.2 almost final)

Once upon a time..

1. A single mechanism for specifying Object-Database Mapping:
 - hibernate .hbm.xml mapping files
2. One Specific Runtime API

Hibernate Today

- Multiple Projects
- Compliance with new EJB3 Persistence Standards
- Supports both xml mapping and Java 5 Annotations
- Supports both the Hibernate API and the EJB3 Persistence API



1. Mapping

- The process of specifying the bindings between an object model and a database schema
- Principal mechanism is via XML mapping files
- *Defacto* file name extension: is .hbm.xml
- Multiple ways to set this up: a single file, one file per class. Best practice is to use one file per class, with each file placed next to its corresponding class file in the package hierarchy, and loaded as a resource

Mapping

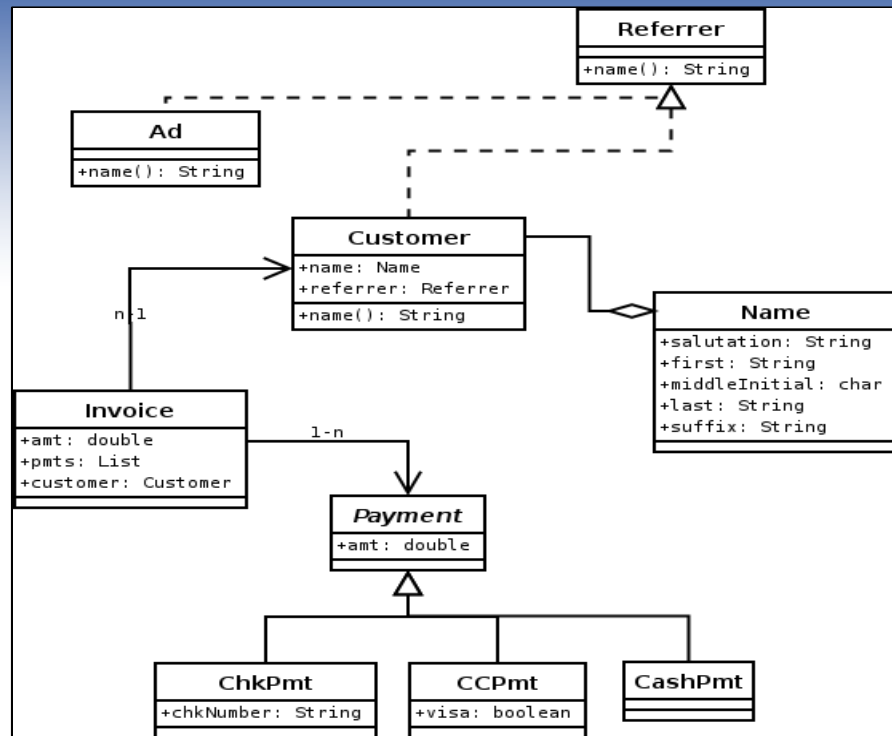
- Entities
- Basic Properties
- Components
- Associations
 - Many-To-One
 - One-To-Many
 - Many-To-Many
- Inheritance Mapping
- Modeling with Interfaces

Model-Centric

- Write Model Classes and Mappings;
Generate Database Schema
- Reverse Engineering Tools available to do the reverse

*Coding a
Sample Application
(live)*

The Model



Types: Entities vs Values

- Analogous to by-reference vs by-value semantics in programming languages
- Just as primitives (int's) are passed by value as parameters to methods (values are copied), by-value semantics in the database implies the same thing: value will map to a table column and will not be shared or referenced by other entities
- Entities on the other hand are the reverse, they are shared (e.g. a many to one scenario)

Components

- Use for giving by-value semantics to Java Classes
- Usage highly encouraged
- Principal mechanism for implementing a “Fine Grained” model (more classes than tables)
- Unlike entities, references are not shared; rather wholly-owned by parent entity; its life-cycle is bound to it
- Components do not get their own tables: they map to columns of their parent's table

Many-One

- Example: Invoice references a single customer. Other invoices may reference that same customer.
- Example mapping:

```
<many-to-one name="customer"  
             column="customer_id" />
```

the column specification references the foreign key in the invoice table

One-Many

- Choose from among the various Collection API types (List, Set, Map, etc..)
- Hibernate also models *Bag* (no implied order) semantics, using a *java.util.List* since the collection API does not provide a Bag type

One-Many: lazy loading

- Default in hibernate v3
- Hibernate implements lazy loading by providing its own Collection API interface implementations.
 - These implementations don't fetch records from the database until explicitly asked for (with a `list.get(i)` for example)
- Consequence: must specify Collection API interfaces in your code (i.e. use List, not ArrayList; otherwise will get a `ClassCastException`)

One-Many *(continued)*

- Example:

```
<bag name="pmts">  
  <key column="invoice_id"/>  
  <one-to-many class="com.u2d.nfjs.Payment"/>  
</bag>
```

- key is foreign key in payment table
- pmts is list property name
- keyword bag is one of a number of choices, including list, set, map

Many-Many

- Many-many associations are specified using an extension of one-many.

- Example:

```
<bag name="actors" table="Movie_Actor">  
  <key column="movies_id"/>  
  <many-to-many column="actors_id"  
    class="com.u2d.movielib.Actor"/>  
</bag>
```

Inheritance

- Four Strategies:
 - Table per class hierarchy
 - Table per subclass
 - Table per concrete class using union-subclass
 - Table per concrete class using implicit polymorphism

Implicit Polymorphism

- Personally a great fan of implicit polymorphism;
- I find this mechanism gives me the freedom to model using interfaces without complicating or sacrificing persistence
- many-to-one associations to polymorphic types specified in mapping file using the <any> tag
- many-to-many associations to polymorphic types specified in mapping file using the <many-to-any> tag

2. The API

- Basic Usage
- What Spring Offers
- Queries
 - HQL (Hibernate Query Language)
 - Criteria API

Basic Usage

Primary Types are:

- SessionFactory
- Session
- Query
- Criteria

Basic Usage: SessionFactory

- One per database
- A factory for sessions
- Container for JVM-level cache (second-level cache)

Prototypical SessionFactory Configuration

```
public class HBMUtil {
    Configuration cfg; SessionFactory factory;

    public HBMUtil()
    {
        cfg = new Configuration();
        cfg.addClass(Customer.class);
        cfg.addClass(Invoice.class);
        // ...
        cfg.setProperty(
            Environment.CURRENT_SESSION_CONTEXT_CLASS,
            "thread");

        factory = cfg.buildSessionFactory();
    }
    ...
}
```

Prototypical Session Interaction

```
Session s = factory.getCurrentSession();
s.beginTransaction();

// interact with session in this "pseudo" block
// for example:
Customer c = new Customer("Eitan");
c.setAccountNo(12345);
s.save(c);

s.getTransaction().commit();
```

What Spring does for Hibernate

- It refactors the use of Hibernate
 - Avoiding duplication of session and transaction setup and teardown code
- Provides various utility methods for common usages
- Provides two implementations:
 - HibernateTemplate / Callback Pattern
 - HibernateInterceptor (a Spring AOP MethodInterceptor)

Spring for Hibernate Example

```
getHibernateTemplate().execute(new HibernateCallback()
{
    public Object doInHibernate(Session session)
    {
        Customer c = new Customer("Eitan");
        c.setAccountNo(12345);
        s.save(c);
    }
})

getHibernateTemplate().fetch("from Customer");
```

Powerful Query Capabilities

- HQL: The Hibernate Query Language
 - object-oriented
- Criteria API
 - powerful object model for constructing and executing queries
- Query by Example
- Not locked in: can perform SQL queries, including stored procedure invocations

HQL

- Powerful object-based query language
- Hibernate translates HQL to SQL
- HQL statements are shorter, more readable than their SQL counterparts

Prototypical Use of Query API

```
String hql = "from Customer c where c.age > :age";  
Query q = session.createQuery();  
q.setInteger("age", 33);  
q.setFirstResult(20);  
q.setMaxResults(10); // fetch the third page  
List customers = q.list(hql);
```

Criteria Queries

- What makes the Criteria API powerful is that it allows queries to be specified by composition.
- This means that queries can be constructed dynamically.

Prototypical Use of Criteria API

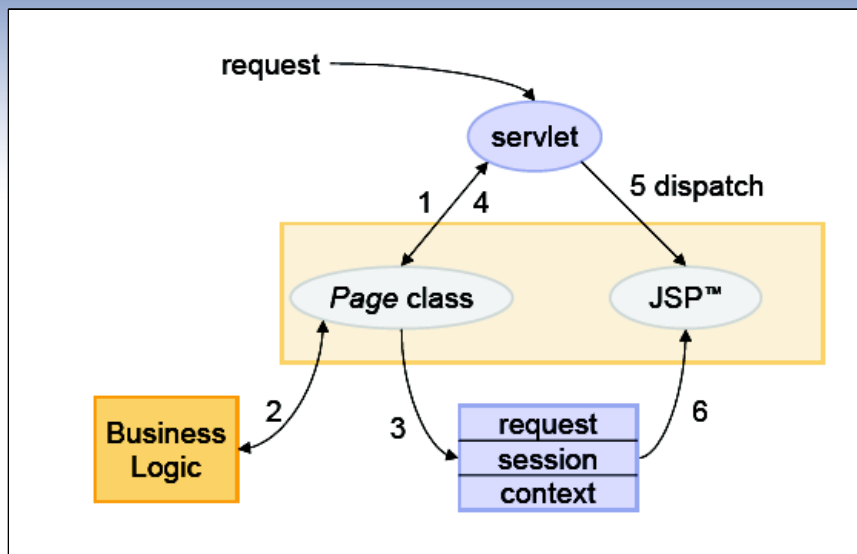
```
Criteria c = session.createCriteria(Customer.class);
c.add( Restrictions.ilike("name", "Albert%") );
c.addOrder( Order.asc("age") );
c.setMaxResults(20);
c.list();

// entire sequence of calls can also be chained,
// like so:
session.createCriteria(Customer.class).
    add( Restrictions.ilike("name", "Albert%") ).
    addOrder( Order.asc("age") ).
    setMaxResults(20).
    list();
```

3. Session Strategies

- Session per request with detached objects
 - a new session is obtained for every request. any objects needed in long conversations must be attached to the new session
- Open Session in View
 - session scope is extended to include view rendering phase
- Session per conversation
 - use same session, but disconnect from underlying JDBC connection after committing a transaction

Hibernate and the Web



Hibernate & Web

- Most Java Web Frameworks provide a Servlet filter that will automatically setup and teardown Hibernate sessions
- Our code can simply fetch the session from the web session or application context, and not worry about having to close the session
- Alternatively, since in MVC all requests go through the same controller, you could put that code directly in the controller servlets. Then all your action classes are all set to interface with a Hibernate session

Hibernate & Web

"*Open Session in View*" Strategy may be convenient for ensuring that view template (JSP et al) doesn't *fault* on lazy-loaded associations after the session has been closed

4. Performance

- Lazy Fetching is a double-edged sword
 - Good to stop cascading fetches ad infinitum
 - Bad if have to perform multiple selects to get a single batch of data for corresponding to a single unit of work (multiple trips across the network)
 - Usually dealt with by specifying default fetch strategy as lazy in mapping files, while performing Eager (now named Join) fetches where appropriate in the code, on a per use-case basis

N+1 Problem Illustration

```
67 public void testCriteriaQuery()
68 {
69     Criteria c = session.createCriteria(Invoice.class);
70     c.setFetchMode("payments", FetchMode.JOIN);
71     List invoices = c.list();
72
73     for (Iterator itr = invoices.iterator(); itr.hasNext(); )
74     {
75         Invoice invoice = (Invoice) itr.next();
76         Set pmts = invoice.getPayments();
77         for (Iterator itr2 = pmts.iterator(); itr2.hasNext(); )
78         {
79             Payment pmt = (Payment) itr2.next();
80         }
81     }
82 }
```

5. Batch Processing

- When using the Hibernate API to insert many records into a database table, the main concerns are:
 - inserted objects are not automatically pushed to the database;
 - *Session* caches the objects
- Remedy is simply to periodically
 - push the changes to the database with a call to *flush()*, and
 - clear the cache with a call to *clear()*

Batch Processing

- Example:

```
Transaction tx = session.beginTransaction();
int i=0;
List<Widget> lotsOfWidgets = loadLotsOfWidgets();

for (Widget widget : lotsOfWidgets)
{
    session.save(widget);

    if ( ((i++) % 20) == 0)
    {
        s.flush();
        s.clear();
    }
}
session.getTransaction().commit();
```

6. UserType

- Can provide your own serialization and deserialization mechanisms for properties
 1. Implement the *UserType* interface
 2. Specify the property type in the mapping using `type="classname"`
 3. Alternatively can create alias for classname with `<typedef>`

UserType Example: TimeSpan

Mapping File:

```
..
<property name="timeSpan"
  type="com.u2d.persist.type.TimeSpanUserType">
  <column name="startDate"
    index="Session_startDate_idx"/>
  <column name="endDate"
    index="Session_endDate_idx"/>
</property>
..
```

Alternatively..

```
..
<typedef name="spantype"
  class="com.u2d.persist.type.TimeSpanUserType" />
<property name="timeSpan" type="spantype">
..
```

UserType Example: TimeSpan

```
public class TimeSpanUserType implements CompositeUserType
{
    public Object nullSafeGet(java.sql.ResultSet rs, String[] names,
        SessionImplementor session, Object owner) ..
    {
        Date from =
            (Date) Hibernate.TIMESTAMP.nullSafeGet(rs, names[0]);
        Date to = (Date) Hibernate.TIMESTAMP.nullSafeGet(rs, names[1]);
        return new TimeSpan(from, to);
    }

    public void nullSafeSet(java.sql.PreparedStatement pstmt,
        Object value, int index, SessionImplementor session)
    {
        TimeSpan span = (TimeSpan) value;
        Hibernate.TIMESTAMP.nullSafeSet(pstmt, span.startDate(), index);
        Hibernate.TIMESTAMP.nullSafeSet(pstmt, span.endDate(),
            index + 1);
    }
    ..
}
```

UserType Example: TimeSpan

```
..
public static final int[] TYPES =
    { java.sql.Types.TIMESTAMP, java.sql.Types.TIMESTAMP };
public int[] sqlTypes() { return TYPES; }
public static String[] COLUMNNAMES = {"startDate", "endDate"};
public String[] getPropertyNames() {
    return new String[] {"start", "end"};
}
public Type[] getPropertyTypes() {
    return new Type[] { Hibernate.TIMESTAMP, Hibernate.TIMESTAMP };
}
public Object getPropertyValue(Object component, int property)
{
    TimeSpan span = (TimeSpan) component;
    if (property == 0) return span.startDate();
    else return span.endDate();
}
public void setPropertyValue(Object component,
    int property, Object value) {
    ..
}
..
```

7. Annotations

- An alternative interface to the Hibernate Core for specifying Mapping
- An alternative to using xml mapping files
- Complies with Annotations Part of EJB3 Persistence Specification

Sample Annotated Class

```
11  @Entity
12  public class Customer
13  {
14      private Long id;
15      @Id @GeneratedValue
16      public Long getId() { return id; }
17      public void setId(Long id) { this.id = id; }
18
19      private Name name;
20      private Referrer referrer;
21
22      public Customer() {}
23      public Customer(String firstname, String lastname)
24      {
25          setName(new Name(firstname, lastname));
26      }
27
28      @Embedded
29      public Name getName() { return name; }
30      public void setName(Name name) { this.name = name; }
31
32      public String toString() { return name.toString(); }
33
34      @Transient
35      public Referrer getReferrer() { return referrer; }
36      public void setReferrer(Referrer referrer)
37      {
38          this.referrer = referrer;
39      }
40  }
```

Sample Annotated Class

```
13 @Entity
14 public class Invoice
15 {
16     private Long id;
17     @Id @GeneratedValue
18     public Long getId() { return id; }
19     public void setId(Long id) { this.id = id; }
20
21     private double amt;
22     private Customer customer;
23     private Set<Payment> payments = new HashSet<Payment>();
24
25     public Invoice() {}
26     public Invoice(double amt) { setAmt(amt); }
27
28     public double getAmt() { return amt; }
29     public void setAmt(double amt) { this.amt = amt; }
30
31     public String toString() { return String.format("Invoice: %f", amt); }
32
33     @ManyToOne
34     public Customer getCustomer() { return customer; }
35     public void setCustomer(Customer customer) { this.customer = customer; }
36
37     @OneToMany
38     public Set<Payment> getPayments() { return payments; }
39     public void setPayments(Set<Payment> payments) { this.payments = payments; }
40     public void addPayment(Payment pmt) { payments.add(pmt); }
41 }
```

8. Tools

- Ant Tools
 - DDL-Related: SchemaExport and SchemaUpdate
- Eclipse Plug-ins
 - Console: HQL scratch pad
 - Mapping Editor
 - Hibernate Configuration File Generator Wizard
 - Reverse Engineering Wizards
 - Custom Hibernate Eclipse “Perspective”

Some Interesting Version 3 Features

- Filters
- XML Entity Mode

Filters

- A simple mechanism to filter tables, similar to what views provide, without having to specify the filter in queries
- Filter can be defined and named in the mapping file
- Filter must be enabled programmatically on a per-session basis with
`session.enableFilter(filterName)`

XML/DOM4J Entity Mode

- A new, Experimental Feature in Hibernate 3
- Very promising, potentially enabling powerful features including import/export, SOAP, and XSLT-based reporting
- Consists of:
 - Adding XML data binding information to mapping files
 - The ability to define a specific entity mode to use when working with a session (for XML, use the DOM4J entity mode)
 - Using the session API to bind database information directly to XML, bypassing object model entirely; bi-directional.

XML Entity Mode

- To interface with Hibernate in this mode:

```
Session session =  
    HibernateUtil.getSessionFactory().openSession();  
  
Session domsession =  
    session.getSession(EntityMode.DOM4J);
```

The Code..

```
19  @Test public void checkOutDom4JMode(Session session) throws Exception
20  {
21      List results = session.createQuery("from Person").list();
22      Document doc = DocumentHelper.createDocument();
23
24      for (Iterator itr = results.iterator(); itr.hasNext(); )
25      {
26          doc.add((Element) itr.next());
27      }
28
29      // write out the document to stdout..
30
31      OutputFormat format = OutputFormat.createPrettyPrint();
32      XMLWriter w = new XMLWriter(System.out, format);
33      w.write(doc);
34      w.close();
35  }
```

The Output

```
<?xml version="1.0" encoding="UTF-8"?>
<person id="1" age="25">
  <first-name>Eitan</first-name>
  <last-name>Suez</last-name>
  <events/>
  <email-addresses/>
</person>
```

XML Mapping Information

```
6 <hibernate-mapping package="com.u2d.runner.hbm_eg">
7   <class name="Event" node="event">
8     <id name="id" node="@id">
9       <generator class="native" />
10    </id>
11    <property name="date" type="timestamp" column="event_date" node="@date" />
12    <property name="title" node="title" />
13
14    <set name="participants" table="registrations" inverse="true" node="participants">
15      <key column="event_id" />
16      <many-to-many column="person_id" class="Person" node="person" embed-xml="false" />
17    </set>
18  </class>
19 </hibernate-mapping>
```

Interesting Observations

- Many O/R mapping solutions have been devised over the years. Hibernate is probably the most successful.
- Effectively addresses major object mapping problems head-on, giving us choices for modeling inheritance, polymorphism
- Flexible framework, can provide own implementations for serializing properties (UserType), how properties are accessed (PropertyAccessor), and more

Conclusions

- Hibernate is a mature, complete solution for addressing Object/Relational mapping
- It is an active project with a large community, large-scale adoption, keeps up with (and has assisted in redefining) Java Persistence Standards and evolution
- Lots of tools: XDoclet / Ant / Eclipse Tooling

References

- <http://www.hibernate.org/>
- *Hibernate In Action* (Bauer & King)
- *Hibernate, a Developer's Notebook* (Elliott)
- *Hibernate Quickly* (Peak & Heudecker)
- *Hibernate* (Iverson)

Contact Information

- Eitan Suez
- <http://u2d.com/>
- email: eitan@u2d.com